Refine Search

Search Results -

Terms	Documents
700/18	353

Database: US Pre-Grant Publication Full-Text Database; US Patents Full-Text Database; US OCR Full-Text Database; EPO Abstracts Database; JPO Abstracts Database; Derwent World Patents Index; IBM Technical Disclosure Bulletins

Search: L5

Refine Search | Recall Text | Clear | Interrupt

Search History

DATE: Wednesday, May 05, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=NO; OP=OR			
L5	700/18	353	L5
L4	(object adj oriented adj3 program$4) and (physical adj5 address) and (equipment)	163	L4
L3	(object adj oriented adj3 program$4) and (physical adj location adj5 address) and (equipment)	3	L3
L2	5923903.pn.	2	L2
L1	(process adj5 program$4 same equipment) and (plural$3 adj3 object) and (execut$3 adj program)	2	L1

END OF SEARCH HISTORY

h e b b cg b e e ch

First Hit

☐ Generate Collection Print

L1: Entry 1 of 2 File: PGPB May 22, 2003

DOCUMENT-IDENTIFIER: US 20030095514 A1
TITLE: Communication device and communication method network system and robot apparatus

Summary of Invention Paragraph:
[0016] A program preparation method according to the present invention comprises: a program preparation step of carrying out radio or wired communication using communication means between an information processing process of an electronic equipment when the electronic equipment controlled by the information processing process is being operated and information processing process on an information processing device, and using data transmitted from the information processing process of the electronic equipment and preparing an information processing process used for the electronic equipment, at the information processing device; and a program embedding step of embedding the information processing process prepared at the program preparation step into the electronic equipment.

Detail Description Paragraph:
[0214] A specific example of the network system using the gateway object will now be described. FIG. 24 shows an exemplary structure of robot software (software including a plurality of object groups) for executing a so-called soccer dog on the basis of the above-described framework for inter-object communication.

Detail Description Paragraph:
[0222] After the trouble is adjusted and the development finally ends, the source code of the SoccerDog object on the remote system 30 is re-compiled for the environment of the target system (robot apparatus 1) and is linked with the existing objects, thus preparing an execution program for the robot apparatus 1. The execution program is introduced to the robot apparatus 1. The robot apparatus 1 has its operation controlled by the execution program thus introduced thereto.

Detail Description Paragraph:
[0240] The program preparation method according to the present invention comprises: a program preparation step of carrying out radio or wired communication using communication means between an information processing process of an electronic equipment when the electronic equipment controlled by the information processing process is being operated and information processing process on an information processing device, and using data transmitted from the information processing process of the electronic equipment and preparing an information processing process used for the electronic equipment, at the information processing device; and a program embedding step of embedding the information processing process prepared at the program preparation step into the electronic equipment. Thus, the information processing device prepares the information processing process in consideration of the electronic equipment existing in the real environment. The electronic equipment has embedded therein such an information processing process prepared in consideration of the real environment, and operates on the basis of the information processing process. Since the information processing process is prepared in consideration of the real environment, the electronic equipment operates without having any trouble.

CLAIMS:

h e b b g e e e f c e e ge

20. A program preparation method comprising: a program preparation step of carrying out radio or wired communication using communication means between an information processing process of an electronic equipment when the electronic equipment controlled by the information processing process is being operated and information processing process on an information processing device, and using data transmitted from the information processing process of the electronic equipment and preparing an information processing process used for the electronic equipment, at the information processing device; and a program embedding step of embedding the information processing process prepared at the program preparation step into the electronic equipment.

h e b b g e e e f c e e ge

First Hit Fwd Refs
End of Result Set

☐ Generate Collection Print

L1: Entry 2 of 2 File: USPT Nov 26, 1996

DOCUMENT-IDENTIFIER: US 5579309 A
TITLE: Object oriented program-controlled broadband communication equipment for optimized method calls

Abstract Text (1):
A switching-oriented process sequencing according to the principle of object-oriented program includes a plurality of object-related structure elements for the implementation of different switching-oriented functions. Objects for the realization of specific switching-oriented functions are instanced by these structure elements, these objects communicating with one another by method calls. It is thereby provided that the network layer for the subscriber signaling defined according to the OSI-7 layer model is subdivided into a plurality of sub-layers. Each of these sub-layers is realized by a structure element hierarchy including at least one structure element. The calling and called subscriber side defined for the subscriber signaling is represented in each structure element of the respective structure element hierarchy by at least one separate object. Objects of one and the same subscriber side thereby communicate directly with one another via quasi-asynchronous method calls, whereas objects of the respective sub-layer of different subscriber sides communicate directly with one another via asynchronous internal events.

Brief Summary Text (5):
With respect to object orientation, this represents a solution that is not optimum because some objects execute methods without influencing the data contained in them merely so that the call chain is not interrupted. It is thus an object of the present invention to disclose a method for reducing the number of method calls for an external signal in a program-controlled equipment. In the program-controlled equipment of the present invention, the complexity of the individual structure elements within the switching-oriented process is reduced and the throughput of the program-controlled equipment is enhanced. This is particularly critical for a broadband ISDN communication equipment.

Detailed Description Text (3):
FIG. 1 schematically illustrates the class structure of the switching-oriented process set forth below. As already disclosed in European patent application EP 0 589 249 A2, a program module--(a switching-oriented program module in this case--) can be subdivided into four layers corresponding to the objects provided therein. The four layers are respectively referred to as trigger, controller, responder agent and responder layer. The trigger layer forms the interface of the program module with the operating system and converts external events into information that can be internally processed. The controller layer contains the switching-oriented flow of control logic, whereas the data base organization of all data required for the execution of the program module is undertaken in the responder agent layer and in the responder layer. Allowable communication relationships between individual objects can thereby be defined, as set forth in the above-referenced European patent application.

CLAIMS:

h e b b g e e e f c e e ge

1. A communication equipment comprising:

means for controlling the communication equipment with at least one object-oriented switching process, the at least one switching process comprising a plurality of object-related structure elements for implementing different switching-oriented functions respectively available for a plurality of switching procedures, said means for controlling further comprising means for instantiating objects by the structure elements for realizing specific switching functions and wherein the objects communicate with one another by method calls, wherein a network layer for subscriber signaling is comprised of a plurality of sub-layers; each of the sub-layers having a structure element hierarchy comprising at least one structure element; and wherein a calling subscriber side and a called subscriber side are represented in every structure element of the respective structure element hierarchy by at least one separate object; and wherein objects of a same subscriber side communicate directly with one another via select ones of said method calls and objects from different subscriber sides communicate directly with one another via asynchronous internal events.

L4: Entry 95 of 163 File: USPT Oct 29, 2002

DOCUMENT-IDENTIFIER: US 6473824 B1
TITLE: Dynamic association of input/output device with application programs

Brief Summary Text (8):
Another possibility is to connect the IO device to the computer system via a fieldbus. Several fieldbus systems have developed into well-known standards. One example is the fieldbus Interbus-S of the PHOENIX company, which is an industrial standard bus for connecting analog and digital devices to control applications in a manufacturing environment. There exist various adapter cards for different hardware (PLC, IPC, etc.). Another example is the fieldbus Profibus of the Siemens company. It is used in industrial manufacturing for controlling machine tools, robots, and other kinds of manufacturing equipment. Another standard system, the fieldbus CAN, is a very fast realtime bus system. The period of time until an event is taken care of can be made very short. Therefore, this fast fieldbus is applied in vehicles; for example, for the control of intelligent braking systems like ABS. In France, there exists a standard for fieldbusses called FIP.

Brief Summary Text (23):
Furtheron, the principles of object-oriented program design are fully obeyed with this approach. While some prior art approaches violated the principle of encapsulation, here this principle is obeyed to.

Drawing Description Text (8):
FIG. 7 shows the communication between a physical object and the configurator object, which provides the physical object with a pointer to its responsible IO Handler, and with the physical address of its IO device.

Detailed Description Text (51):
Whenever an IO domain object of any subclass of the base IOPhysicalObject class 500 is instantiated, it obtains a unique logical address. This address is wrapped in an address object (IOAddress, 520). There exist several subclasses (522, 523, 524) to the base IOAddress class, which implement different address formats. The respective address object is possessed (519) by the physical object IOPhysicalObject (500), which means that the address object is completely contained in the physical object.

Detailed Description Text (52):
In FIG. 6, the interactions of a multitude of physical objects and IO Handlers are shown. For each device that has to be controlled, a new physical object with a new logical address is instantiated. The corresponding IO Handlers, for example the Interbus-S, may be shared among different physical objects. Therefore, the corresponding IO Handler only has to be instantiated in case it doesn't exist yet. It is not necessary to provide multiple copies of one and the same IO Handler.

Detailed Description Text (53):
PhysicalObject 0 (600) is the physical object with the logical address 0. It possesses a pointer for establishing a communication link (601) with the Interbus-S DeviceDriver 602. Physical Object 0 (600) controls the lamp 603, which is connected to the fieldbus Interbus-S, and which has the physical address "lamp III".

Detailed Description Text (54):
In FIG. 6, two more physical objects (608, 611) are shown that also communicate with the Interbus-S DeviceDriver 602. Each of these physical objects addresses a different IO device having a different physical address. Physical Object 2 (608) communicates (609) with the Interbus-S DeviceDriver 602 and controls lamp 610, which has the physical address "lamp I." Physical Object 3 (611) controls (612) the lamp 613 with the physical address "lamp II".

Detailed Description Text (55):
The two remaining physical objects, Physical Object 1 (604) and Physical Object 4 (614), control devices that are attached to the COM port. Therefore, they communicate (605, 615) with the IO Handler COM_Port Access (606). Physical Object 1 (604) controls device 607 with the physical address "device B," and Physical Object 4 (614) controls device 616 with the physical address "device A".

Detailed Description Text (56):
For each device, there exists a physical object with a unique logical address that controls said device. This physical object only communicates with the IO Handler the corresponding device is attached to. This IO Handler is responsible for all the devices attached to the IO interface controlled by the IO Handler. Therefore the IO Handler may be shared among different physical objects. The physical object's logical address is mapped to one specific IO Handler and to a specific physical address of a device controlled by said IO Handler. The following table summarizes the assignment of logical addresses to IO Handlers and physical device addresses for the example given in FIG. 6.

Detailed Description Text (57):
The information of this table is the content of the so-called resource file, which has to be specified by the user. Besides the resource file, there exists a class of configurator objects which are responsible for providing each Physical Object with its IO Handler, and with the physical address of its IO device. They are responsible for the administration of the assignments given in the resource file.

Detailed Description Text (59):
When a Physical Object is instantiated, it is given its logical address. FIG. 7 shows the steps of the routine "translateLogicalToReal," which is responsible for providing a Physical Object with the pointer to its IO Handler, and with the physical address of its IO device.

Detailed Description Text (60):
In the first step of said routine, the IO Physical Object (700) sends a message "getIOConfigurator" (701), together with its logical address, to the Domain Configurator (702). The Domain configurator, which is the root class of the configurator class tree, knows which IO Configurator is responsible for the Physical Object 700. It sends a "responsible" message (703), together with the logical address, to the responsible IO Configurator 704. In the next step, "returnHandleToIOConfigurator" (705), the IO Configurator provides the IO Physical Object with a pointer to its IO Configurator. Therefore, the IO Physical Object may now address its IO Configurator directly.

Detailed Description Text (62):
The IO Physical Object (700) possesses an object of the type IOAddress. So far, said address object contains the IO Physical Object's logical address. In the next step, "setPhysicalAddress-AndIOHandlerType" (708), this logical address contained in the Physical Object is overwritten with the physical address. This implies that the step of address translation is only required once. Future accesses of the IO Physical Object to the IO Handler can take place without an address translation, because from now on the IO Physical Object knows the physical address of its IO device. In the next step, "returnHandleToIOHandler" (709), the IO Physical Object (700) obtains a pointer to its corresponding IO Handler 706. Now the IO Physical

Object (700) can directly access its IO Handler (706) by means of an "object reference", because it possesses both the pointer to said IO Handler and the physical address of the IO device to be controlled. Step 710 shows an example of an access of the IO Physical Object to the IO Handler 706. The IO Physical Object may read data from ("readIOData") or write data to ("writeIOData") the associated IO Handler, and thus to the associated IO device. It may also send control messages to the IO device ("controlDevice"). The communication path between the IO Physical Object (700) and the IO Handler (706) is now established. The IO Configurator (704) has fulfilled its task and is not required any more. As soon as the user intends to link a different IO Handler with the IO Physical Object 700, though, the IO Configurator is required again. It has to provide the IO Physical Object with a pointer to the new IO Handler and with a new physical address. Such a change of the configuration is possible at runtime.

Detailed Description Text (64):
Let us assume that the IO Configurator 806 corresponds to the COM Port. All Physical Objects that control IO devices attached to the COM Port forward their logical addresses to the IO Configurator 806 and obtain their physical addresses from there. For example, the IO Physical Object 807 possesses (808) an object of the type IOAddress (809), which at first contains the Physical Object's logical address. Next, the routine of FIG. 7 is carried out. In FIG. 8, the whole routine of FIG. 7 is referred to as "translateLogicalToRea" (8IO). In the step "setPhysicalAddress-AndIOHandlerType" (708) of FIG. 7, the content of the IO Address object 809 is overwritten with the physical address.

Detailed Description Text (65):
The IO Configurator (806) is responsible for mapping the logical addresses to the physical addresses. For this purpose, the IO Configurator possesses (811) a list (812) of address maps. Said list (811) possesses (813) N objects of the type "IOAddressMaps" (814), whereby each of said address maps comprises (815, 817) two objects of the type "IOAddress" (816, 818). One of these objects "IOAddress" (818) contains the logical address and the second one (816) contains the corresponding physical address. Initially, the assignment of logical addresses to both IO Handlers and physical addresses is only contained in the resource file. This resource file is parsed by the object "IODidoParser" (803), and therefore, this parser has to "fill" (804) the IO Address objects (816, 818) of the various Address Maps with the required logical and physical addresses.

Detailed Description Paragraph Table (1):
Logical Address IO Handler Physical Address Physical Object 0 Interbus-S lamp III Physical Object 1 COM Port device B Physical Object 2 Interbus-S lamp I Physical Object 3 Interbus-S lamp II Physical Object 4 COM Port device A

CLAIMS:

3. An object-oriented framework according to claim 1 or claim 2, said framework further comprising a configurator object, for providing, after having obtained a physical object's logical address, said physical object with a pointer to the corresponding IO device driver and with the physical address of the IO device that said physical object is supposed to control.

5. A method for establishing a communication path according to claim 4 wherein said step of establishing an object reference between said physical object and said device driver comprising the steps of: forwarding said physical object's logical address from said physical object to a configurator object; and returning, from said configurator object to said physical object, a pointer to said IO device driver and a physical address of said IO device that said physical object is supposed to control.

L4: Entry 97 of 163 File: USPT Sep 17, 2002

DOCUMENT-IDENTIFIER: US 6453460 B1
TITLE: Computer system with single processing environment for executing multiple application programs

Brief Summary Text (2):
Embodiments of the present invention relate to methods of developing object oriented programs and to systems that execute object oriented programs.

Brief Summary Text (4):
An object oriented program is a computer program that has been written in a computer programming language having semantics for describing software components designed to cooperate by passing messages between them. Such a computer program defines the overall program behavior as a combination of the behavior of so-called "objects". An object is an association of data structure and behavior. An object oriented program may employ multitasking program behavior.

Brief Summary Text (5):
Although the computer that performs an object oriented program may have only one central processing unit for the execution of only one instruction at a time, some of the instructions of each object may be executed in turn in slices according to known techniques of time sharing.

Brief Summary Text (6):
An object oriented program may appear to be responsive to a wider range of inputs than a conventional non-object oriented program for providing the same overall behavior. Programmers have found that for complex behavior, a program developed using object oriented programming techniques is more likely to provide correct behavior in response to a particular sequence of inputs from a wide range of inputs too numerous to test in all combinations.

Brief Summary Text (7):
The semantics of an object oriented programming language permit the description of a program having multiple objects, each operating with its own values. For example, a display of multiple circles being continuously and seemingly simultaneously redrawn at random times with random positions and sizes may be defined by an object oriented program with multiple objects, each object having the function to draw one circle, and each operating with exclusive access to its own variables. As seen by the central processing unit (CPU), the behavior of an object is expressed as a sequence of instructions in the executable instruction code appropriate for the CPU fetched from specific addresses in memory. Such instructions cause the CPU to read and modify the values of numerous variables at specific addresses in the memory. To achieve seemingly simultaneous redrawing of circles, execution of each object is restricted to a time slice. Because a time slice may lapse at any moment during the sequence being performed by one of the circle drawing objects, data describing the position in the instruction sequence (a thread) must be saved and later referred to by the CPU to continue a particular circle when that particular object is given another time slice.

Brief Summary Text (8):
Each object has behavior and state. The functions (i.e., operations or services) of

an object are called its behavior. The data describing values of variables and the data describing the position in the instruction sequence are collectively called the object's state. An object oriented program defines in a program specification the behavior of each unique object and the variables needed for its functions. This definition is, in some programming languages, called a class from which one or more instances of the object may be set in motion (e.g., instantiated).

Brief Summary Text (13):
A memory, according to various aspects of the present invention, has indicia of an object oriented program that in operation provides indicia of a state. The state includes two instances of a factory and two instances of an object. The first instance of the factory includes a first instance variable. The second instance of the factory includes a second instance variable. The first instance variable is responsive to either the instantiation of the first object by the factory or responsive to an operation of the first object. The second instance variable is responsive to either the instantiation of the second object by the second factory or responsive to an operation of the second object. The first instance variable is independent of both the instantiation of the second object and the operation of the second object.

Brief Summary Text (15):
A printer, according to various aspects of the present invention, includes a print engine, a processor, and a memory. The processor provides data to the print engine. The memory has indicia of an object oriented program executed by the processor to provide indicia of a state as discussed above. When, for example, the first and second instance variables of the respective factories respond respectively to operations of the objects, each object may use its factory's instance variable independently of the operations of the other object. As another example, when the execution environment includes a JAVA virtual machine (JVM) that supports a single processing space, independent execution as described above permits multiple application programs to independently use the behavior of the same library object with independent states.

Brief Summary Text (17):
The revision to the first specification and the criteria for revising (or initially writing) application specifications provides for concurrent application execution in a manner that is considerably simpler than, for example, modifying the execution environment to permit multiple separate processing spaces. The capability to perform multiple application programs is supported with no change to the processing environment. For example, when applied to the specifications of object oriented programs to be executed by a JVM as discussed above, the resulting integration will operate on one unmodified JVM intended for single process execution. The expense of providing a JVM for each application specification may be avoided.

Detailed Description Text (2):
A computer system, according to various aspects of the present invention, obtains independent operation of multiple application programs in a single processing space. Such a computer system includes any conventional circuitry or network of equipment that provides an object oriented program execution environment. Automated equipment (e.g., a printer) may be part of such a computer system and may internally incorporate such a computer system. Generally, an execution environment is provided by a conventional operating system. For executing platform independent, object oriented programs, the processing environment may be provided by a conventional virtual machine or interpreter (e.g., a JAVA Virtual Machine, JAVA being a trademark of Sun Microsystems Inc.). For example, a computer system 100 as in FIG. 1 or a printer 500 as in FIG. 5 may each provide an execution environment 200 as in FIG. 2. A computer system 100 may include any number of processors 120, any number of memories 130, and any number of input/output subsystems 150 coupled together for data transfer and control by bus 140.

Detailed Description Text (3):
Memory, according to various aspects of the present invention, includes indicia of multiple program specifications and, in operation, includes data structures for recording the state of objects instantiated from the program specifications. For example, specifications in some programming languages are called "Classes" and may be arranged in one or more data structures to exhibit hierarchical relationships between program specifications. Physically, memory 130, 518 may include any combination of conventional data storage and retrieval devices, including, for example, disk memory or semiconductor memory. Memory may be integrated with one or more processors on a single substrate, or may be packaged as a circuit module for convenient storage of information, handling, and incorporation into computer systems and/or automated equipment. For example, a JVM may be provided in one or more physically separate packages (e.g., integrated circuit substrates), chip set (s), or module(s)) and application programs to be executed by the JVM may be provided in the same or other packages. Different organizations may contribute package(s) for incorporation into computer system 100 or printer 500 by the original equipment manufacturer or by a technician or user in the field.

Detailed Description Text (5):
An executive process includes any process responsible for platform specific behaviors including power-on initialization, device, bus, and memory circuit control, device related error conditions, and power-off procedures. For example, executive 202 includes all procedures necessary for preparing for execution and executing a conventional program specification of an object oriented program.

Detailed Description Text (14):
Note that the relevant contents of memory 130 are described logically in FIG. 4 without reference to physical aspects of memory addresses. The logical layout is convenient for illustration. A physical memory map of the functions of various address ranges indeed may have any intermixed configuration convenient for processing. For example, compiled and linked code blocks for the instructions 202, 212, 222, 232, and 242 may result in any arrangement of fragmented address ranges; and, the allocation of ranges of addresses for variables 402, tables 262, 264, 266, and processing space 268 may also result in any arrangement of fragmented address ranges. Memory 130 may support any function (instruction block, table, etc.) with multiple physically discontiguous ranges. As memory is allocated for additional environment variables, additions to tables, and additions to processing space, for example, further fragmentation may result without departing from the logical description illustrated in FIG. 4 or the various functional aspects of the present invention. Further, any physical layout of the contents of memory 130 may include the effects of access limitations as in segmented memory, extended memory, expanded memory, demand paging from disk memory, swap tables, shared memory for parallel processors, caches, etc.

Detailed Description Text (23):
Automated rquipment, according to various aspects of the present invention, may include a computer system as described above. Some examples of such equipment include computers, computer peripherals, telecommunications equipment, process control equipment, and instrumentation. For example, printer 500 of FIG. 5 includes processor 514 and memory 518 coupled by bus 512 in a manner generally corresponding to processor 120, memory 130, and bus 140 of FIG. 1. Printer 500 also includes controls and displays 516, I/O interface 510, and print engine 520 each coupled to bus 512 and collectively generally corresponding to input/output subsystems 150.

CLAIMS:

1. An object oriented program executed in an execution environment, the execution environment including a memory having indicia of the object oriented program and data structures for recording indicia of a state of objects instantiated during the object oriented program operation, the state comprising: a. a first instance of a

factory, the first instance comprising a first instance variable; b. a second instance of the factory, the second instance comprising a second instance variable; c. a first instance of an object, the first instance variable being responsive to a first operation of a first set consisting of forming the first instance of the object by the first instance of the factory and an operation of the object with the first instance of the object; and d. a second instance of the object, the second instance variable being responsive to a second operation of a second set consisting of forming the second instance of the object by the second instance of the factory and an operation of the object with the second instance of the object; wherein e. the first instance variable is independent of the second operation.

2. The object oriented program of claim 1 wherein the indicia of the object oriented program comprises indicia of a class hierarchy.

3. The object oriented program of claim 2 wherein the hierarchy comprises: a. a first subtree comprising a first specification that, in operation, directs the first instance of the factory to form the first instance of the object; and b. a second subtree comprising a second specification that, in operation, directs the second instance of the factory to form the second instance of the object.

4. The object oriented program of claim 3 wherein the hierarchy further comprises a specification of the factory and a specification of the object.

5. The object oriented program of claim 1 wherein the memory further includes indicia of a virtual machine for performing the object oriented program.

6. The object oriented program of claim 1 wherein the indicia of the object oriented program comprises JAVA byte code.

7. A printer comprising: a. a print engine; b. a processor that provides data to the print engine; and c. a memory having indicia of an object oriented program and data structures for recording indicia of a state of objects, the object oriented program executed by the processor to provide indicia of the state, the state comprising: (1) a first instance of a factory, the first instance comprising a first instance variable; (2) a second instance of the factory, the second instance comprising a second instance variable; (3) a first instance of an object, the first instance variable being responsive to a first operation of a first set consisting of forming the first instance of the object by the first instance of the factory and an operation of the object with the first instance of the object; and (4) a second instance of the object, the second instance variable being responsive to a second operation of a second set consisting of forming the second instance of the object by the second instance of the factory and an operation of the object with the second instance of the object; wherein (5) the first instance variable is independent of the second operation.

8. The printer of claim 7 wherein the indicia of the object oriented program comprises indicia of a class hierarchy.

L4: Entry 109 of 163 File: USPT Feb 5, 2002

DOCUMENT-IDENTIFIER: US 6345351 B1
TITLE: Maintenance of speculative state of parallel executed jobs in an information processing system

Detailed Description Text (10):
The system memory 107, which is accessible to each processor 103, has a predefined physical memory address range. In order to access (i.e., read or write) any location within the system memory 107, it is therefore necessary to present a corresponding physical memory address to the system memory 107 by means of the memory bus 111. However, computer programs (e.g., operating system, application programs, emulation programs, etc.) running on any of the processors 103 do not use these physical addresses directly. Instead, each computer program sees a virtual address space that may or may not be the same size as that of the physical address space.

Detailed Description Text (11):
Because of the use of virtual addressing, it is necessary to translate each virtual memory address into a physical memory address whenever the system memory 107 is to be accessed. To facilitate this process, the virtual and physical address ranges are each divided into blocks of contiguous addresses. Each one of these blocks, which is also known as a "page", has a fixed number of consecutive memory addresses associated with it. Typically, each page has a size of 2 N, where N is an integer. Thus, given a pointer to the start of a page (in either the virtual or physical memory space), an N-bit offset address may be used to access any location within the page.

Detailed Description Text (12):
Each page is, itself, associated with a unique page number that distinguishes it from all other pages. Assuming, then, that the number of pages in the memory (whether virtual or physical) is 2 M, an address that uniquely identifies one memory location can be formed by concatenating the page number with the N-bit offset address described above to form an M+N bit address. As mentioned earlier, the size of the virtual memory space need not be equal to the size of the physical memory space. Consequently, the value of "M" need not be the same for a virtual address as it is for a physical address.

Detailed Description Text (13):
As mentioned earlier, the M+N bit virtual addresses used by the running programs need to be translated into physical addresses before they can be supplied to the system memory 107. To perform this function in the exemplary embodiment, each processor 103 is equipped with a memory management unit (MMU) 119 that treats the most significant M bits of each address as a page number, and the remaining N bits as an offset within the page. A page table (not shown in FIG. 1) located within the system memory 107 uniquely maps each of the 2 M virtual pages to a corresponding physical page in the system memory 107. When the MMU 119 performs logical to physical address translation for each memory read or write, it does so by performing a table look-up (also referred to as a "table walk"), locating the relevant page table entry, and then calculating the physical address. The traditional way to organize a page table is as a two- or three-level indexed look-up table, or as a hash table. To speed up page table look-up, special caches,

called Translation Look-aside Buffers (TLBs) are introduced for holding the most-used translations. When a system includes TLBs, table look-up is needed only when the TLB fails to include a translation for a requested virtual page. This occurrence, which is referred to as a "TLB miss", typically causes an interrupt that not only performs the necessary table lookup, but also loads an appropriate entry in the TLB so that this translation can be more efficiently performed in the future.

Detailed Description Text (18):
Memory management is typically transparent to ordinary application programs. Consequently, a program's view of how its storage is laid out in virtual address space (virtual storage) need not match how that storage is arranged in the physical address space (physical storage). In particular, the system memory 107 may appear to the program as a sequence of consecutive memory addresses (in virtual storage), even though it may actually be mapped to a number of scattered (e.g., non-contiguous and/or out of sequence) physical memory pages within the system memory 107. This permits the underlying operating system to utilize memory allocation strategies that make most efficient use of the available physical memory.

Detailed Description Text (21):
To provide this adaptation, the application program 201 interfaces with an emulator program 203. Emulator programs are well-known in the art as programs that create a virtual machine that allows applications, originally written for another processor, to execute on an available, different processor either by interpreting the application code or by recompiling the application code. For example, a Java Virtual Machine (JVM) is an emulator that allows Java byte code to execute on almost any processor. Emulator programs may either directly interface with the underlying processing equipment, or may alternatively operate by means of interaction with an operating system 205 (shown in dotted lines in FIG. 2) that directly interacts with the underlying processing equipment.

Detailed Description Text (22):
In the present environment, the emulator program 203 (either alone, or acting in conjunction with the operating system 205) creates a virtual machine that allows the application program 201 to execute on the available multi-processor system 101. Here, one of the tasks of the emulator 203 is to transform the application program 201 into one or more jobs 207, each of which will be run on one of the processors 103 in the system 101. Separate jobs can be created for basic blocks in the application program 201, for individual iterations in a loop, for method calls in an object oriented program or between tasks. The particular approach taken in this regard is up to the designer, and is beyond the scope of this description.

Detailed Description Text (46):
The job 207 then executes using the pages associated with the shared memory (step 507). This situation is illustrated in the block diagram of FIG. 6a. In this example, two jobs 207 (numbered "1" and "2") share a virtual memory space, and execute concurrently. When each of the jobs 207 accesses a virtual page N, the access is mapped to an original shared physical page, here illustrated by shared physical page P. However, the second job 207 has made a backup 603 of shared physical page P. The backup 603, which is located in physical page Q, is not mapped to any of the virtual pages within the shared virtual address space. Consequently, the backup 603 is not part of the shared physical address space.

Detailed Description Text (55):
The job 207 then executes using the strategy outlined above. This situation is illustrated in the block diagram of FIG. 8a. In this example, two jobs 207 (numbered "1" and "2") share a virtual memory space, and execute concurrently. When the first job 209 accesses virtual page N, it is mapped to a shared physical page P in the physical memory 801. However, the second job 207 in this example is generating speculative data to virtual page N. Consequently, the second job's

accesses to virtual page N are now mapped to a private physical page Q. The private physical page Q is not mapped to any of the virtual pages within the shared virtual address space. Consequently, the private physical page Q is not part of the shared physical address space. Also illustrated in FIG. 8a is an example in which each of the first and second jobs 207 have their accesses mapped to a same shared physical page S if neither is generating speculative data for a shared virtual page R.

Detailed Description Text (58):
This situation is illustrated in the block diagram of FIG. 8c. Here, a subsequent job 207 (denoted job "3") has its accesses to virtual page N mapped to what is now the shared physical page Q. The physical page P that was formerly part of the shared physical address space is now an available physical page.

Detailed Description Text (85):
This exemplary system may further use PCBs, rather than signals, to maintain a starting state for the job until it is known that the job will not need to be restarted (i.e., the job has been run and retired). A suitable PCB for this purpose is illustrated in FIG. 11. The exemplary PCB 1101 includes a portion for maintaining the initial state 1103, that is, processor register contents to be loaded when the job starts plus other administrative information. The PCB 1101 further includes a first pointer 1105 to a shared page table 1107 and a second pointer 1109 which points to a private page table 1111. The shared page table 1107 can, itself, comprise multiple levels of tables. For example, the first pointer 1105 to the shared page table may actually point to a first table 1113. Each entry contained within the first table 1113 is a pointer 1115 to a corresponding second table 1117, which contains the actual physical memory addresses and other information. In accordance with this architecture, mapping a virtual address to a physical address requires several lookup steps.

L4: Entry 132 of 163 File: USPT Nov 9, 1999

DOCUMENT-IDENTIFIER: US 5980078 A
TITLE: Process control system including automatic sensing and automatic configuration of devices

Abstract Text (1):
A digital control system with a predetermined configuration automatically senses the connection to a network of a digital device that is not included in the predetermined configuration. The digital device is assigned temporary address information and placed in a temporary state, called a standby state, in which the digital device supplies information to the digital control system allowing a user to access the digital device including access of device information and configuration parameters. Using the device information and configuration parameters, a user selectively commissions the digital device by assigning a physical device tag, a device address, and a device identification, and installing a control strategy to the digital device, thereby placing the digital device in an operational state in communication with the digital control system. In the standby state, a user interrogates to determine the type of device that is attached, determines the role of the device in the context of the digital control system, assigns a physical device tag that assigns the determined role to the device, and verifies connection of the device to the network. Also in the standby state, the user initiates other applications applied to the device, including calibration of the device and configuring the device within the overall control scheme of the digital control system.

Brief Summary Text (5):
Present-day process control systems use instruments, control devices and communication systems to monitor and manipulate control elements, such as valves and switches, to maintain at selected target values one or more process variables, including temperature, pressure, flow and the Eke. The process variables are selected and controlled to achieve a desired process objective, such as attaining the safe and efficient operation of machines and equipment utilized in the process. Process control systems have widespread application in the automation of industrial processes such as the processes used in chemical, petroleum, and manufacturing industries, for example.

Brief Summary Text (15):
In accordance with a further aspect of the present invention, a digital control system with a predetermined configuration automatically senses the connection to a network of a digital device that is not included in the predetermined configuration. The digital device is assigned temporary address information and placed in a temporary state, called a standby state, in which the digital device supplies information to the digital control system allowing a user to access the digital device including access of device information and configuration parameters. Using the device information and configuration parameters, a user selectively commissions the digital device by assigning a physical device tag, and a device address, and installing a control strategy to the digital device, thereby placing the digital device in an operational state in communication with the digital control system. In the standby state, a user interrogates to determine the type of device that is attached, determines the role of the device in the context of the digital control system assigns a physical device tag that assigns the determined

role to the device, and verifies connection of the device to the network. Also in the standby state, the user initiates other applications applied to the device, including calibration of the device and configuring the device within the overall control scheme of the digital control system.

Detailed Description Text (2):
Referring to FIG. 1, a front-of-screen display, also called a "screen" 100, for a graphical user interface (GUI) depicts a display of a system configuration. The screen 100 depicts navigation selections which are operated by a user to select, construct and operate a process control configuration. The navigation program supplies an initial state for navigating across various tools and processors in a network. A user controls the navigation program to access libraries, areas, process control equipment and security operations.

Detailed Description Text (36):
The operation of commissioning a new device 300 results in a condition in which the device is assigned a new physical device tag and a device address, and the device is ready for function block configuration. The new field device is entered into the process control system network database with the device identification bound and the device revision information set. An engineering software tool that displays the process control system network status displays the new device as a COMMISSIONED device.

Detailed Description Text (40):
In step 320, a new user assigns a new physical device tag to the field device. The physical device tag is constrained to be unique and not the same as the device identification. During the assignment of the physical device tag, a device address is assigned to the field device using a software engineering tool and the Link-Active-Scheduler takeover preference is set to "selectable". The device revision information is read from the field device and written to the process control system network database. The interface changes the state of the field device 322 to the INITIALIZED state using a clear address service (CLEAR-ADDRESS). The field device appears in the "live list" at a temporary address 324.

Detailed Description Text (42):
Referring to FIG. 4, a flow chart illustrates a second operation or "use case" which describes an operation of commissioning an unbound device 400. Prior to the commissioning of the unbound device, the Fieldbus interface is operational. A field device has been created in the process control system network database and a physical device tag and a device address are assigned to the field device. However, the field device is not bound to a device identification. The process control system network database has also been initialized to contain device revision information read from the field device. A software engineering tool displays the field device as an UNBOUND device. The UNBOUND device to be commissioned is either a field device with no physical device tag or a field device having a physical device tag that is identical to the device identification. The UNBOUND field device is commissioned to place the field device on the process control system network link.

Detailed Description Text (43):
The operation of commissioning an UNBOUND device 400 results in a condition in which the device is configured with a physical device tag and an assigned device address, and the device is ready for function block configuration. The new field device is entered into the process control system network database with the device identification bound. An engineering software tool that displays the process control system network status displays the device as a COMMISSIONED device.

Detailed Description Text (92):
The process control environment 1100 is implemented using an object-oriented framework. An object-oriented framework uses object-oriented concepts such as class

hierarchies, object states and object behavior. These concepts, which are briefly discussed below, are well known in the art. Additionally, an object-oriented framework may be written using object-oriented programming languages, such as the C++ programming language, which are well-known in the art, or may be written, as is the case with the preferred embodiment, using a non-object programming language such as C and implementing an object-oriented framework in that language.

Detailed Description Text (101):
The process control environment 1100 includes multiple subsystems with several of the subsystems having both a configuration and a run-time implementation. For example, a process graphic subsystem 1230 supplies user-defined views and operator interfacing to the architecture of the process control environment 1100. The process graphic subsystem 1230 has a process graphic editor 1232, a part of the configuration implementation 1210, and a process graphic viewer 1234, a portion of the run-time implementation 1220. The process graphic editor 1232 is connected to the process graphic viewer 1234 by an intersubsystem interface 1236 in the download language. The process control environment 1100 also includes a control subsystem 1240 which configures and installs control modules and equipment modules in a definition and module editor 1242 and which executes the control modules and the equipment modules in a run-time controller 1244. The definition and module editor 1242 operates within the configuration implementation 1210 and the run-time controller 1244 operates within the run-time implementation 1220 to supply continuous and sequencing control functions. The definition and module editor 1242 is connected to the run-time controller 1244 by an intersubsystem interface 1246 in the download language. The multiple subsystems are interconnected by a subsystem interface 1250.

Detailed Description Text (104):
The run-time implementation 1220 interfaces to the persistent database 1270 and to local databases 1262 to access data structures formed by the configuration implementation 1210. In particular, the run-time implementation 1220 fetches selected equipment modules, displays and the like from the local databases 1262 and the persistent database 1270. The run-time implementation 1220 interfaces to other subsystems to install definitions, thereby installing objects that are used to create instances, when the definitions do not yet exist, instantiating run-time instances, and transferring information from various source to destination objects.

Detailed Description Text (107):
The Explorers 1310 is operated by a user to select, construct and operate a configuration. In addition, the Explorer.TM. 1310 supplies an initial state for navigating across various tools and processors in a network. A user controls the Explorer.TM. 1310 to access libraries, areas, process control equipment and security operations. FIG. 13 illustrates the relationship between various tools that may be accessed by a task operating within the process control environment 1100 and the relationship between components of the process control environment 1100 such as libraries, areas, process control equipment and security. For example, when a user selects a "show tags" function from within an area, a "tag list builder" is displayed, showing a list of control and I/O flags. From the tag list builder, the user can use an "add tag" function to add a module to a list, thereby invoking a "module editor".

Detailed Description Text (108):
Referring to FIG. 14, a schematic block diagram illustrates a hierarchical relationship among system objects of a configuration model 1400. The configuration model 1400 includes many configuration aspects including control, I/O, process graphics, process equipment, alarms, history and events. The configuration model 1400 also includes a device description and network topology layout.

Detailed Description Text (109):

The configuration model hierarchy 1400 is defined for usage by a particular set of users for visualizing system object relationships and locations and for communicating or navigating maintenance information among various system objects. For example, one configuration model hierarchy 1400, specifically a physical plant hierarchy, is defined for usage by maintenance engineers and technicians for visualizing physical plant relationships and locations and for communicating or navigating maintenance information among various instruments and equipment in a physical plant. An embodiment of a configuration model hierarchy 1400 that forms a physical plant hierarchy supports a subset of the SP88 physical equipment standard hierarchy and includes a configuration model site 1410, one or more physical plant areas 1420, equipment modules 1430 and control modules 1440.

Detailed Description Text (110):
The configuration model hierarchy 1400 is defined for a single process site 1410 which is divided into one or more named physical plant areas 1420 that are defined within the configuration model hierarchy 1400. The physical plant areas 1420 optionally contain tagged modules, each of which is uniquely instantiated within the configuration model hierarchy 1400. A physical plant area 1420 optionally contains one or more equipment modules 1430. An equipment module 1430 optionally contains other equipment modules 1430, control modules 1440 and function blocks. An equipment module 1430 includes and is controlled by a control template that is created according to one of a number of different graphical process control programming languages including continuous function block, ladder logic, or sequential function charting ("SFC"). The configuration model hierarchy 1400 optionally contains one or more control modules 1440. A control module 1440 is contained in an object such as a physical plant area 1420, an equipment module 1430 or another control module 1440. A control module 1440 optionally contains objects such as other control modules 1440 or function blocks. The control module 1440 is thus a container class, having instances which are collections of other objects. The control module 444 is encapsulated so that all of the contents and the implementation of the methods of the control module are hidden.

CLAIMS:

3. A process control system comprising:

a process;

a plurality of devices coupled to the process;

a communication network coupled to the devices;

a workstation coupled to the plurality of devices via the network and including a user interface; and

a software system executable on the network and implementing a routine for automatically sensing a connection of a device to a network and placing the connected device in an accessible state for communicating with a user via the user interface, the software system including:

a routine for commissioning the connected device including:

a user-interactive routine for assigning a physical device tag, a device address, and a device identification to the connected device; and

a user-interactive routine for installing a control strategy to the digital device.

6. A control system comprising:

a network;

a plurality of devices coupled to the network;

a distributed controller coupled to the plurality of devices and controlling the plurality of devices according to a defined control configuration, the distributed controller including:

a control logic for sensing a device that is connected to the network but not included in the defined control configuration;

a control logic for supplying initial interconnect information to the connected device;

a control logic for uploading configuration parameters from the connected device to the distributed controller; and

a control logic for commissioning the connected device including:

a user-interactive control logic for assigning a physical device tag, a device address, and a device identification to the connected device; and

a user-interactive control logic for installing a control strategy to the digital device.

7. A method of configuring a control system comprising:

predetermining a configuration of devices coupled to a network;

sensing a connection to the network of a device that is not included in the predetermined configuration;

assigning the connected device a standby address which allows access to device information and configuration parameters of the connected device;

commissioning the connected device into an operational state in communication with the control system, including:

assigning to the connected device a physical device tag, a device address, and a device identification;

installing a control strategy to the connected device; and

placing the connected device in an operational state in communication with the network; and

configuring the connected device in combination with the predetermined configuration of devices.

16. An executable logic operating in a network for configuring a control system comprising:

means for predetermining a configuration of devices coupled to a network;

means for sensing a connection to the network of a device that is not included in the predetermined configuration;

means for assigning the connected device a standby address which allows access to device information and configuration parameters of the connected device;

means for commissioning the connected device into an operational state in communication with the control system, including:

means for assigning to the connected device a physical device tag, a device address, and a device identification;

means for installing a control strategy to the connected device; and

means for placing the connected device in an operational state in communication with the network; and

means for configuring the connected device in combination with the predetermined configuration of devices.